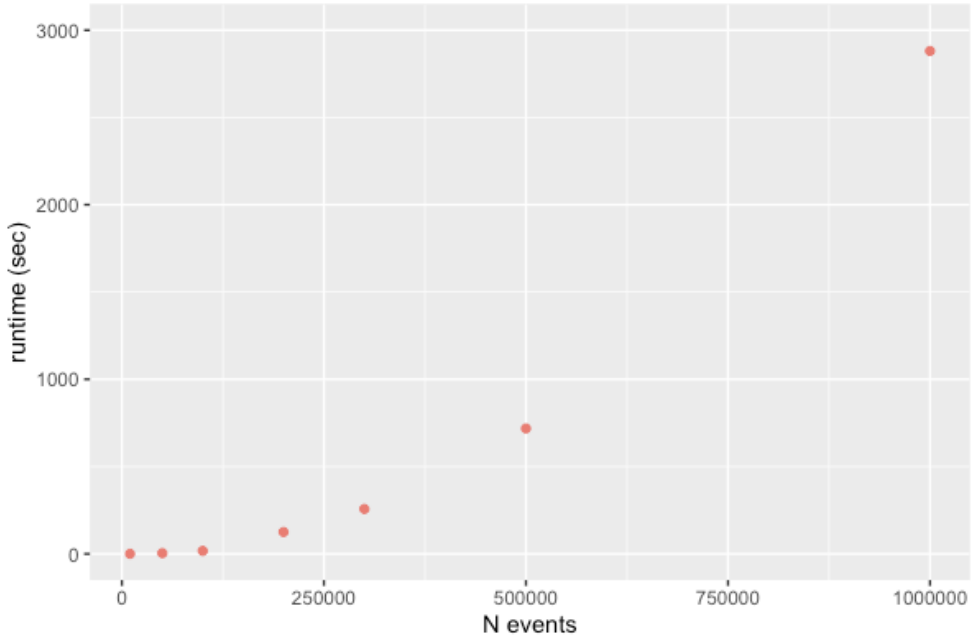
Jamie Goodin
12/17/22
SST Challenge Algorithm Rationale

Starting off (Method 1)

When first approaching this problem, I sorted the events using nested `for` loops. For every event, it iterated through each existing level until it found a match (or not). However, this did not scale well.



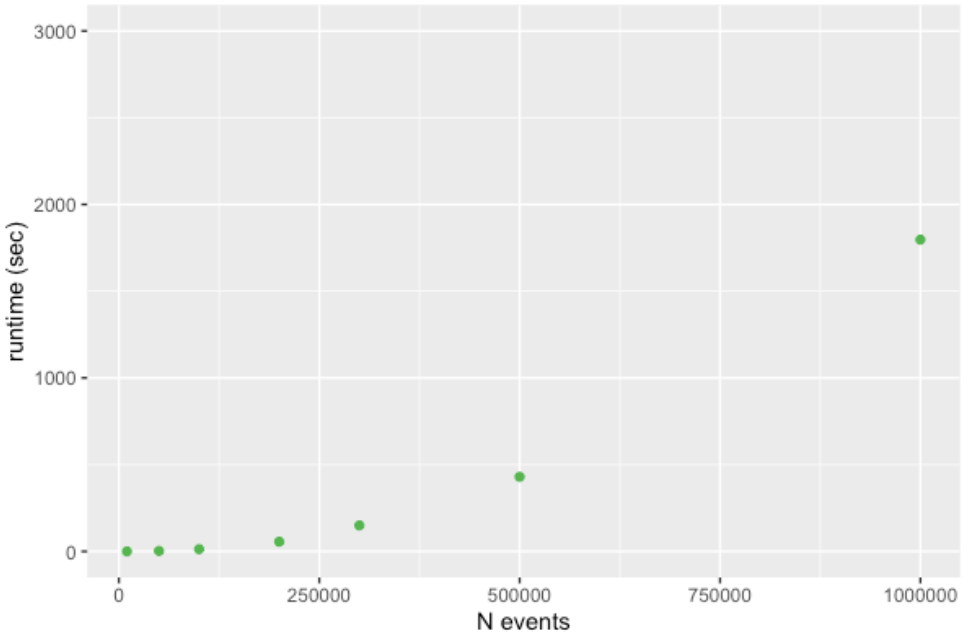
Method 1 performance

Interval tree

To try and improve the algorithm, I used interval trees to represent each level to leverage O(n*log n) insertion and intersection query time. In the end, this did not result in improved algorithmic complexity or real world performance.

A level-forward approach (Method 2)

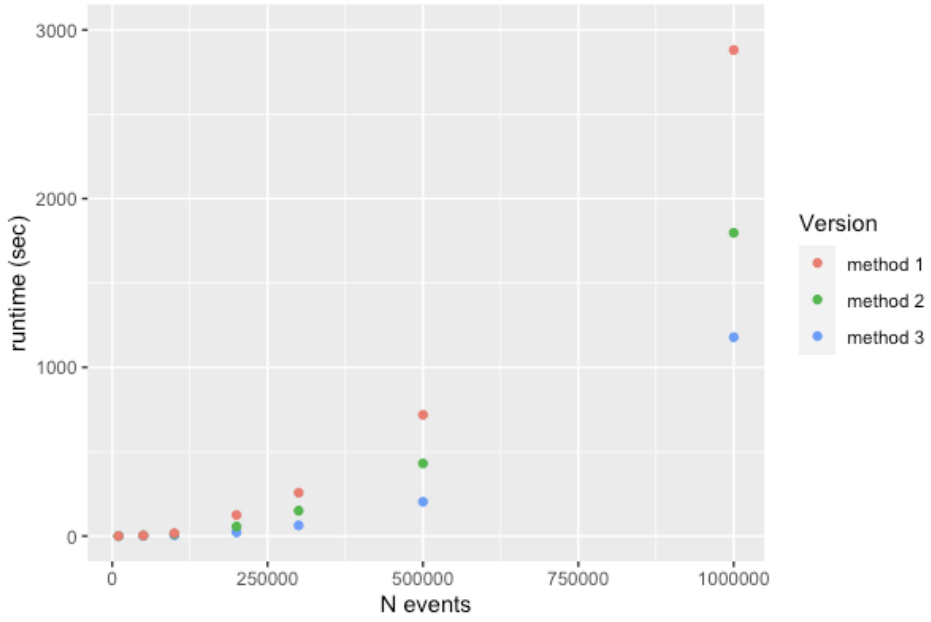
Rather than iterating through each event and searching each level for a fit in a growing pool of levels, the algorithm could instead fill a base level with as many events as possible and create and fill new levels until each event is sorted. This outperformed the first method.



Method 2 performance

Sorted approach (Method 3)

The final method and fastest method sorts the inputs by `endTime`. And compares the first event in the list's `endTime` to the next event's `startTime`. If the next event starts before the first event ends, it pushes it to the level. It cycles through each event making this comparison, resulting in significantly better scaling.



Method 3 performance compared to Methods 1 and 2

Conclusion

After a lot of research and trying a few different methods, I have produced an algorithm that sorts correctly, and much more efficiently than some other options. This was a fun and interesting technical problem to analyze, and it's easy to see how it's relevant to Surgical Safety Technology's products. I look forward to discussing this challenge with your team, and learning more ways to approach it.